An Efficient Implementation of the Thomas-Algorithm for Block Penta-diagonal Systems on Vector Computers

Katharina Benkert[1] and Rudolf Fischer[2]

[1] High Performance Computing Center Stuttgart (HLRS), University of Stuttgart, 70569 Stuttgart, Germany
benkert@hlrs.de
[2] NEC High Performance Computing Europe GmbH, Prinzenallee 11, 40549 Duesseldorf, Germany
rfischer@hpce.nec.com

Abstract. In simulations of supernovae, linear systems of equations with a block penta-diagonal matrix possessing small, dense matrix blocks occur. For an efficient solution, a compact multiplication scheme based on a restructured version of the Thomas algorithm and specifically adapted routines for LU factorization as well as forward and backward substitution are presented. On a NEC SX-8 vector system, runtime could be decreased between 35% and 54% for block sizes varying from 20 to 85 compared to the original code with BLAS and LAPACK routines.

Keywords: Thomas algorithm, vector architecture.

1 Introduction

Neutrino transport and neutrino interactions in dense matter play a crucial role in stellar core collapse, supernova explosions and neutron star formation. The multidimensional neutrino radiation hydrodynamics code PROMETHEUS / VERTEX [1] discretizes the angular moment equations of the Boltzmann equation giving rise to a non-linear algebraic system. It is solved by a Newton Raphson procedure, which in turn requires the solution of multiple block-penta-diagonal linear systems with small, dense matrix blocks in each step. This is achieved by the Thomas algorithm and takes a major part of the overall computing time. Since the code already performs well on vector computers, this kind of architecture has been the focus of the current work.

The Thomas algorithm [2,3] is a simplified form of Gaussian elimination without pivoting, as originally applied to tridiagonal systems. Bieniasz [4] gives a comprehensive overview of the numerous adaptations for special cases and mutations of tridiagonal systems, the extensions to cyclic tridiagonal systems and the transfer to block tridiagonal matrices. However, an efficient implementation for block penta-diagonal systems has not yet been considered in the literature. An additional challenge consists in solving systems with relatively small matrices

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 144–151, 2007.
© Springer-Verlag Berlin Heidelberg 2007

as used in the Thomas algorithm. Optimizations of LAPACK routines usually target large matrices as necessitated for example for the HPL benchmark [5].

The remainder of the paper is organized as follows: after introducing the Thomas algorithm and the concept of vector architectures, section 2 presents the compact multiplication scheme. Section 3 explains implementation issues for the newly introduced scheme and the LU decomposition and states the results obtained, followed by the summary of the paper in section 4.

1.1 Thomas Algorithm

Consider a linear system of equations consisting of a block penta-diagonal (BPD) matrix with n blocks of size $k \times k$ in each column resp. row, a solution vector $\underline{x}$ and a right hand side (RHS) $\underline{f}$. The vectors $\underline{x} = (\underline{x}_1^T \underline{x}_2^T \dots \underline{x}_n^T)^T$ and $\underline{f} = (\underline{f}_1^T \underline{f}_2^T \dots \underline{f}_n^T)^T$ are each of dimension $k \cdot n$. The BPD system is defined by

$$\mathsf{A}_i \underline{x}_{i-2} + \mathsf{B}_i \underline{x}_{i-1} + \mathsf{C}_i \underline{x}_i + \mathsf{D}_i \underline{x}_{i+1} + \mathsf{E}_i \underline{x}_{i+2} = \underline{f}_i, \quad 1 \le i \le n, \tag{1}$$

by setting, for ease of notation, $\mathsf{A}_1 = \mathsf{B}_1 = \mathsf{A}_2 = \mathsf{E}_{n-1} = \mathsf{D}_n = \mathsf{E}_n = \mathsf{0}$, and implementing $\underline{x}$ and $\underline{f}$ as $(\underline{x}_{-1}^T \underline{x}_0^T \dots \underline{x}_n^T)^T$ and $(\underline{f}_{-1}^T \underline{f}_0^T \dots \underline{f}_n^T)^T$.

Eliminating the sub-diagonal matrix blocks A_i and B_i and inverting the diagonal matrix blocks C_i would result in the following system:

$$\begin{array}{rrrl} \underline{x}_i + \mathsf{Y}_i\, \underline{x}_{i+1} + & \mathsf{Z}_i\, & \underline{x}_{i+2} & = \underline{r}_i, \qquad 1 \le i \le n-2, \\ \underline{x}_{n-1} + \mathsf{Y}_{n-1}\, & \underline{x}_n & & = \underline{r}_{n-1}, \\ & \underline{x}_n & & = \underline{r}_n. \end{array} \tag{2}$$

In the Thomas algorithm, the new components Y_i, Z_i and $\underline{r}_i$ are computed by substituting $\underline{x}_{i-2}$ and $\underline{x}_{i-1}$ in (1) using the appropriate equations of (2) and comparing coefficients. This results in

$$\left.\begin{array}{l} \mathsf{Y}_i = \mathsf{G}_i^{-1}\, (\mathsf{D}_i - \mathsf{K}_i \mathsf{Z}_{i-1}) \\ \mathsf{Z}_i = \mathsf{G}_i^{-1}\, \mathsf{E}_i \\ \underline{r}_i = \mathsf{G}_i^{-1}\, (\underline{f}_i - \mathsf{A}_i \underline{r}_{i-2} - \mathsf{K}_i \underline{r}_{i-1}) \end{array}\right\} \quad i = 1, n, \tag{3}$$

where

$$\left.\begin{array}{l} \mathsf{K}_i = \mathsf{B}_i - \mathsf{A}_i \mathsf{Y}_{i-2} \\ \mathsf{G}_i = \mathsf{C}_i - \mathsf{K}_i \mathsf{Y}_{i-1} - \mathsf{A}_i \mathsf{Z}_{i-2} \end{array}\right\} \quad i = 1, n, \tag{4}$$

assuming again, for ease of notation, $\mathsf{Y}_{-1} = \mathsf{Z}_{-1} = \mathsf{Y}_0 = \mathsf{Z}_0 = \mathsf{0}$. Then the solution is computed using backward substitution, i.e.

$$\begin{array}{rrrrl} \underline{x}_n & = \underline{r}_n, & & & \\ \underline{x}_{n-1} & = \underline{r}_{n-1} & - \mathsf{Y}_{n-1}\, \underline{x}_n, & & \\ \underline{x}_i & = \underline{r}_i & - \mathsf{Y}_i\, \underline{x}_{i+1} & - \mathsf{Z}_i \underline{x}_{i+2}, & i = n-2, -1, 1. \end{array} \tag{5}$$

Since in practice, the inversion of G_i in (3) is replaced by a LU-decomposition $\mathsf{G}_i = \mathsf{L}_i \mathsf{U}_i$, it's crucial to understand that solving a BPD system includes two levels of Gaussian elimination (GE) and backward substitution (BS): one for the

whole system and one for each block row. The GE of the whole system (GE_S) requires the calculation of K_i and G_i shown in (4). GE and BS are then applied to each block row (GE_R / BS_R) computing Y_i, Z_i and $\underline{r}_i$ in (3). Finally, backward substitution (5) is applied to the whole system (BS_S) to obtain the solution $\underline{x}$.

1.2 Vector Architecture

Two different kinds of computer architectures are available today: scalar and vector. They use different approaches to tackle the common problem, memory latencies and memory bandwidth. Vector architectures [6,7] use SIMD instructions in particular also for memory access to hide memory latencies. This requires large, independent data sets for efficient calculation. The connection to main memory is considerably faster than for scalar architectures. Since there are no caches, data are directly transfered to and from main memory. Temporary results are stored in vector registers (VRs) of hardware vector length VL, which is 256 on the NEC SX-8 [8,9]. As not all instructions are vectorizable, scalar ones are integrated into vector architectures. Because of their slow execution compared with commodity processors their use should be avoided by improving the vectorization ratio of the code.

2 Reordering the Computational Steps

For operations on small matrix blocks, memory traffic is the limiting factor for total performance. Our approach involves reordering the steps in (3) and (4) as

$$\begin{array}{lll} \mathsf{K}_i = \mathsf{B}_i - \mathsf{A}_i\,\mathsf{Y}_{i-2} & \mathsf{G}_i = \mathsf{G}'_i - \mathsf{K}_i\,\mathsf{Y}_{i-1} & \mathsf{Y}_i = \mathsf{G}_i^{-1} \cdot \mathsf{H}_i \\ \mathsf{G}'_i = \mathsf{C}_i - \mathsf{A}_i\,\mathsf{Z}_{i-2} & \mathsf{H}_i = \mathsf{D}_i - \mathsf{K}_i\,\mathsf{Z}_{i-1} & \mathsf{Z}_i = \mathsf{G}_i^{-1} \cdot \mathsf{E}_i \\ \underline{r}'_i = \underline{f}_i - \mathsf{A}_i\,\underline{r}_{i-2} & \underline{r}''_i = \underline{r}'_i - \mathsf{K}_i\,\underline{r}_{i-1} & \underline{r}_i = \mathsf{G}_i^{-1} \cdot \underline{r}''_i, \end{array} \tag{6}$$

where the following spaces can be shared: B_i and K_i; C_i, G'_i and G_i; D_i, H_i and Y_i; E_i and Z_i, as well as $\underline{f}_i$, $\underline{r}'_i$, $\underline{r}''_i$ and $\underline{r}_i$. The improvements of this rearrangement are valuable for the GE of the whole system as well as for the solution of the block rows. First, during GE_S, A_i and K_i are loaded only k times from main memory instead of $3k$ times as is the case with a straight forward implementation. Second, by storing H_i, E_i and $\underline{r}''_i$ contiguously in memory, the inverse of G is applied simultaneously to a combined matrix of size $k \times (2k+1)$ during GE_R / BS_R. Third, by supplying a specifically adapted own version of LAPACK's xGETRF (factorization) and xGETRS (forward and backward substitution) routines [10], memory traffic is further reduced. By combining factorization and forward substitution, L_i is applied to the RHS vectors during the elimination process and therefore not reloaded from main memory.

3 Implementation Details and Numerical Results

In this section, the performance of our new approach presented in section 2 is compared to the standard approach which uses LAPACK's xGETRF and

xGETRS routines to compute the LU-factorization and apply forward / backward substitution and the BLAS routines xGEMM and xGEMV [11,12] for matrix-matrix products and matrix-vector products, respectively. All computations were carried out on a NEC SX-8 vector computer using the proprietary Fortran90 compiler version 2.0 / Rev. 340 [13].

After presenting the results for the compact multiplication scheme used for GE_S, different implementations of GE_R and BS_R are analyzed. Since the number of RHS vectors is $2k + 1$ and therefore depends on the block size k, a small, a medium and a large test case with block sizes of $k = 20$, 55 and 85 are selected representatively. After that, the performance of the solution process for one block row as well as for the whole system (1) is described.

3.1 Applying Gaussian Elimination to the Whole System

The first two systems of (6) are of the form

$$\begin{aligned} \mathsf{D} &= \mathsf{B} + \mathsf{A} \cdot \mathsf{C}, \\ \mathsf{G} &= \mathsf{E} + \mathsf{A} \cdot \mathsf{F}, \\ \underline{z} &= \underline{v} + \mathsf{A} \cdot \underline{w}. \end{aligned} \tag{7}$$

Using BLAS, each line is split into two operations, one copy operation, e.g. $\mathsf{D} = \mathsf{B}$, and one call to xGEMM or xGEMV. Instead, if A should only "once" be loaded from main memory, the system (7) has to be treated as a single entity and xGEMM or xGEMV can not be used any longer since neither the matrices nor the vectors are stored contiguously in memory.

Our implementation listed below is a compact multiplication scheme. It contains two parts, the first one calculating the first column of D and G as well as the whole vector $\underline{z}$, the second computing the remaining columns of D and G. The loop in i is stripmined to work on vectors smaller or equal to the hardware vector length VL.

```
do is = 1, k, VL
   ie = min( is+VL-1, k )

! start with vector z and first columns of D and G
   save v(is:ie), B(is:ie,1) and E(is:ie,1) to VRs vx, vm1 and vm2
   do l = 1, k
      store w(l), C(l,1) and F(l,1) to scalars val1, val2 and val3
      multiply A(is:ie,l) with val1, val2 and val3 and save
         results to VRs vxt, vm1t and vm2t
      add VRs vxt(is:ie), vm1t(is:ie) and vm2t(is:ie) to VRs
         vx(is:ie), vm1(is:ie) and vm2(is:ie)
   end do
   store VRs vx, vm1 and vm2 to z(is:ie), D(is:ie,1) and G(is:ie,1)

! do rest of columns of D and G
   do j = 2, k
```

```
      save B(is:ie,j) and E(is:ie,j) to VRs vm1 and vm2
      do l = 1, k
         store C(l,j) and F(l,j) to scalars val1 and val2
         multiply A(is:ie,l) with val1 and val2 and save results
            to VRs vm1t and vm2t
         add VRs vm1t(is:ie) and vm2t(is:ie) to VRs vm1(is:ie) and
            vm2(is:ie)
      end do
      store VRs vm1 and vm2 to D(is:ie,j) and G(is:ie,j)
   end do
end do
```

Tbl. 1 compares the performance of the BLAS code to the introduced compact multiplication scheme. The latter leads to an average decrease of $11 - 31\%$ in execution time. It is considerably faster for small and medium block sizes, whereas for large block sizes, its performance is approached by BLAS.

Table 1. Execution time of different implementations solving (7) for 150000 calls

$k =$	20	30	40	50	60	70	80	90
LAPACK [s]	2.23	4.65	7.68	11.97	16.92	22.49	28.44	37.99
comp. mult. [s]	1.54	3.25	5.93	9.07	13.19	18.49	25.28	33.71
decr. in runtime [%]	31.1	30.1	22.9	24.2	22.1	17.8	11.1	11.3

3.2 Applying Gaussian Elimination and Backward Substitution to One Block Row

Gaussian Elimination. The search of the pivot element and exchange of matrix rows is not very costly. Measurements on the NEC SX-8 indicated that the search for the maximum value in the pivot column using Fortran's intrinsic function *maxval* and then looping until the correct index has been found is generally faster than a search over the whole pivot column.

The performance of the different versions of GE is depicted in figure 1, 2 and 3. The simplest implementation of row-oriented GE with immediate update [14] for a given step j consists of just two loops in i (column index) and l (row index) updating the remaining parts of the matrix and RHS vectors. The loop in l over the remaining rows is the longer one and therefore the innermost. This loop order is switched by the compiler which naturally degrades performance. The next variant "val + UR(4)", introducing a temporary scalar for elements of the pivot row, still with the compiler's switching of loops, leads to an unrolling of depth 4 in l causing strided memory access. The compiler directive select(vector) on the SX-8, normally used for parallel constructs, provides a simple means to

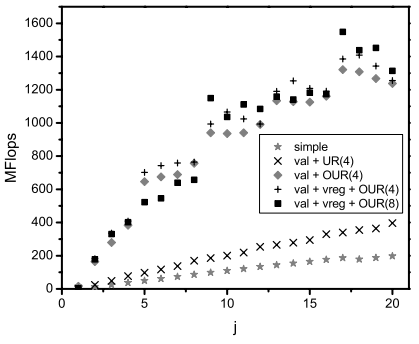

Fig. 1. Performance for GE_R for $k = 20$

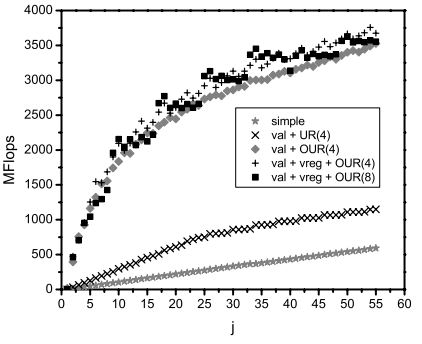

Fig. 2. Performance for GE_R for $k = 55$

enforce the correct order of the loops. This version "val + OUR(4)" with a temporary scalar now has a compiler-generated outer unrolling in i with depth 4. The remaining two variants "val + vreg + OUR(4)" and "val + vreg + OUR(8)" use VRs for the pivot row, a temporary scalar for elements of the pivot column as well as outer unrolling in i with depths 4 or 8. As can be seen from these figures, the correct loop order is essential for good performance. Using temporary scalars for elements of the pivot column, a VR for elements of the pivot row and outer unrolling in i of depth 8 leads essentially to the highest MFlop rates on the tested vector system.

Backward Substitution. The results for the different versions of row-wise backward substitution explained below are shown in figure 4, 5 and 6. The simplest version for a given step j also consists of just two loops. The obvious first improvement, introducing a temporary scalar for elements of the "pivot row" leads to compiler-generated inner unrolling with depth 9993 and is therefore not shown. The second variant "val + OUR(4)" imposes outer unrolling with depth 4. The remaining variants "val + vreg + OUR(8)", "val + vreg + OUR(16)" and "val + vreg + OUR(4/16)" use one VR for elements of the pivot row, temporary scalars for elements of the pivot column as well as outer unrolling of different depths. Apparently, the fastest version uses a VR for elements of the pivot row, temporary scalars for elements of the pivot column and explicitly coded, yet compiler-generated, outer unrolling of depth 16 as long as possible and of depth 4 for the remaining columns.

Performance Results. Using the optimal implementation for GE_R / BS_R, runtimes on the NEC SX-8 may be reduced by the following factors (for 50000 calls) compared to the LAPACK version: by 61.5% from 5.04s to 1.94s for $k = 20$, by 43.3% from 23.98s to 13.59s for $k = 55$ and by 41.2% from 54.54s to 32.08s for $k = 85$.

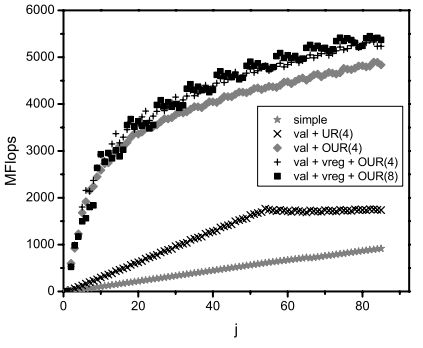

Fig. 3. Performance for GE_R for $k = 85$

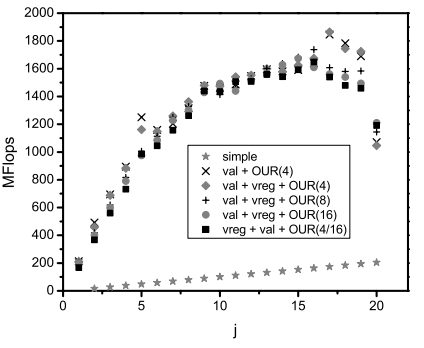

Fig. 4. Performance for BS_R for $k = 20$

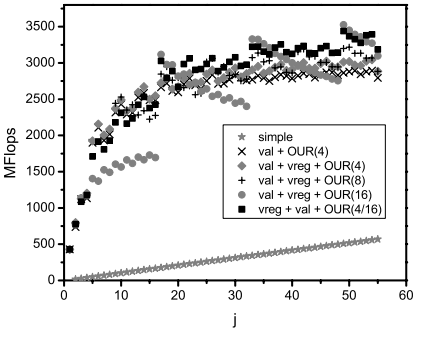

Fig. 5. Performance for BS_R for $k = 55$

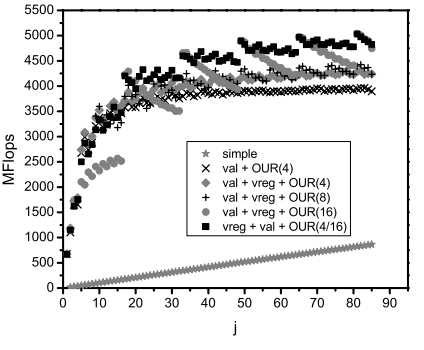

Fig. 6. Performance for BS_R for $k = 85$

3.3 Solution of a Sample BPD System

The compact multiplication scheme and the improvements of the solution of the block rows are integrated into a new BPD solver. Its execution times are compared to a traditional BLAS/LAPACK solver in table 2 for 100 systems with block sizes $k = 20, 55$ and 85 and $n = 500$.

Except for the first version using bandwise storage, the diagonals are stored as five separate vectors of matrix blocks. If H_i, Z_i and $\underline{r}_i$ in (6) are stored contiguously in memory, only one call to xGETRS is needed instead of three.

Table 2. Execution times for BPD solver

$k =$	20	55	85
BLAS/LAPACK + bandw. stor. [s]	11.63	36.79	79.70
BLAS + 3 LAPACK calls [s]	8.33	40.22	85.26
BLAS + 1 LAPACK call ($\star$) [s]	6.43	33.80	76.51
comp. mult. + new solver ($\star\star$) [s]	3.79	23.10	55.10
decr. in runtime between ($\star$) and ($\star\star$) [%]	54.5	42.6	35.4

4 Summary

The solution of a linear system of equations with a BPD matrix with block sizes ranging from 20 to 85 was investigated. A substitute for xGEMM and xGEMV based on a restructured version of the Thomas algorithm as well as specifically adapted versions of LAPACK's xGETRF and xGETRS routines were implemented. Best results were obtained with a compact multiplication routine for the global system and a combined factorization and forward substitution scheme for each block row. The latter uses temporary scalars for elements of the pivot column, a vector register for elements of the pivot row and outer unrolling of depth 8 during Gaussian elimination and of depth 16 and 4 during backward substitution. For a NEC SX-8 vector system a decrease in total runtime between 35% and 54% for test cases of block size 20, 55 and 85 compared to the original code using BLAS and LAPACK was achieved.

An efficient implementation for scalar architectures will be the subject of further research.

References

1. Rampp, M., Janka, H.T.: Radiation hydrodynamics with neutrinos. Astron. Astrophys. **396** (2002) 361–392
2. Thomas, L.H.: Elliptic problems in linear difference equations over a network. Watson Sci. Comput. Lab. Rept., Columbia University, New York (1949)
3. Bruce, G.H., Peaceman, D.W., Jr. Rachford, H.H., Rice, J.D.: Calculations of unsteady-state gas flow through porous media. Petrol. Trans. AIME **198** (1953) 79–92
4. Bieniasz, L.K.: Extension of the Thomas algorithm to a class of algebraic linear equation systems involving quasi-block-tridiagonal matrices with isolated block-pentadiagonal rows, assuming variable block dimensions. Computing **67** (2001) 269–284
5. Dongarra, J.: Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee (1989)
6. Kitagawa, K., Tagaya, S., Hagihara, Y., Kanoh, Y.: A hardware overview of SX-6 and SX-7 supercomputer. NEC Res. & Develop. **44** (2003) 2–7
7. Haan, O.: Vektorrechner: Architektur - Programmierung - Anwendung. Saur, München (1993)
8. Joseph, E., Snell, A., Willard, C.G.: NEC launches next-generation vector supercomputer: The SX-8. IDC # 4290 (2004) 1–15 White Paper.
9. HLRS. `http://www.hlrs.de/hw-access/platforms/sx8/` (2006)
10. Anderson, E., Blackford, L.S., Sorensen, D., eds.: LAPACK User's Guide. Society for Industrial & Applied Mathematics (2000)
11. Dongarra, J.J., Croz, J.D., Hammarling, S., Hanson, R.J.: An extended set of Fortran basic linear algebra subprograms. ACM Trans. Math. Soft. **14** (1988) 1–17
12. Dongarra, J.J., Croz, J.D., Hammarling, S., Duff, I.S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Soft. **16** (1990) 1–17
13. NEC Corporation: SUPER-UX Fortran90/SX Programmer's Guide. Revision 1.2. (2005)
14. Ortega, J.M.: Introduction to parallel and vector solution of linear systems. Frontiers of computer science. Plenum Press, New York (1988)